TUNING FUNDAMENTALS

Basics of Proportional-Integral-Derivative Control

PID controllers are by far the most popular feedback controllers for continuous processes. Here's a look at how they work.

Vance J. VanDoren, *CONTROL ENGINEERING*

A feedback controller is designed to generate an *output* that causes some corrective effort to be applied to a *process* so as to drive a measurable *process variable* towards a desired value known as the *setpoint*. The controller uses an *actuator* to affect the process and a *sensor* to measure the results. Figure 1 shows a typical feedback control system with blocks representing the dynamic elements of the system and arrows representing the flow of information, generally in the form of electrical signals.

Virtually all feedback controllers determine their output by observing the *error* between the setpoint and a measurement of the process variable. Errors occur when an operator changes the setpoint intentionally or when a process *load* changes the process variable accidentally.

In warm weather, a home thermostat is a familiar controller that attempts to correct temperature of the air inside a house. It measures the room temperature with a thermocouple and activates the air conditioner whenever an occupant lowers the desired room temperature or a random heat source raises the actual room temperature. In this example, the house is the process, the actual room temperature inside the house is the process variable, the desired room temperature is the setpoint, the thermocouple is the sensor, the activation signal to the air conditioner is the controller output, the air conditioner itself is the actuator, and the random heat sources (such as sunshine and warm bodies) constitute the loads on the process.

PID control

A proportional-integral-derivative or PID controller performs much the same function as a thermostat but with a more elaborate algorithm for determining its output. It looks at the current value of the error, the integral of the error over a recent time interval, and the current derivative of the error signal to determine not only how much of a correction to apply, but for how long. Those three quantities are each multiplied by a *tuning constant* and added together to produce the current controller output CO(t) as in equation [1]. In this equation, P is the *proportional* tuning constant, I is the *integral* tuning constant, D is the *derivative* tuning constant, and the error e(t) is the difference between the setpoint P(t) and the process variable PV(t) at time t. If the current error is large or the error has been sustained for

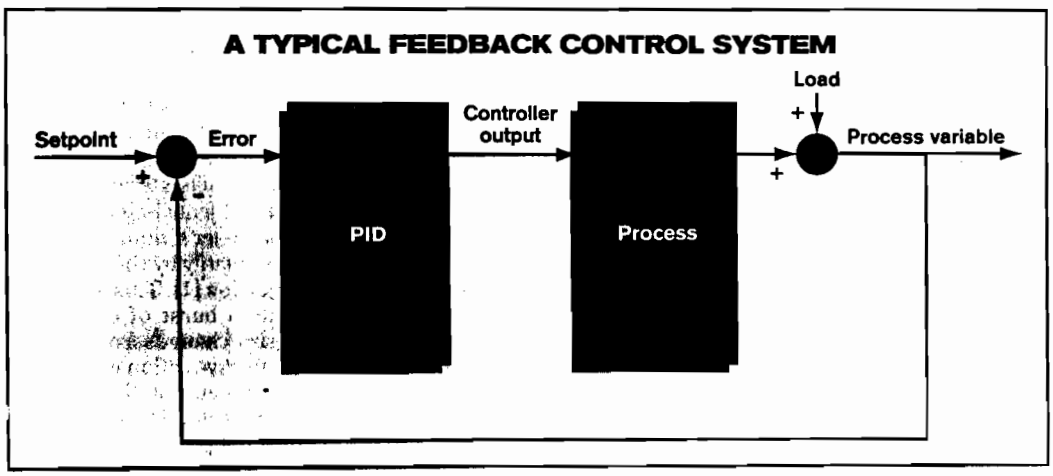

Fig. 1: Most feedback controllers for continuous processes use the proportional-derivative-integral (PID) algorithm to manipulate the process variable by applying a corrective effort to the process.

CONTROL ENGINEERING

KEY WORDS

- Process control & instrumentation
- Process control systems
- PID (Proportional/Integral/Derivative)
- Controllers

P is the proportional tuning constant

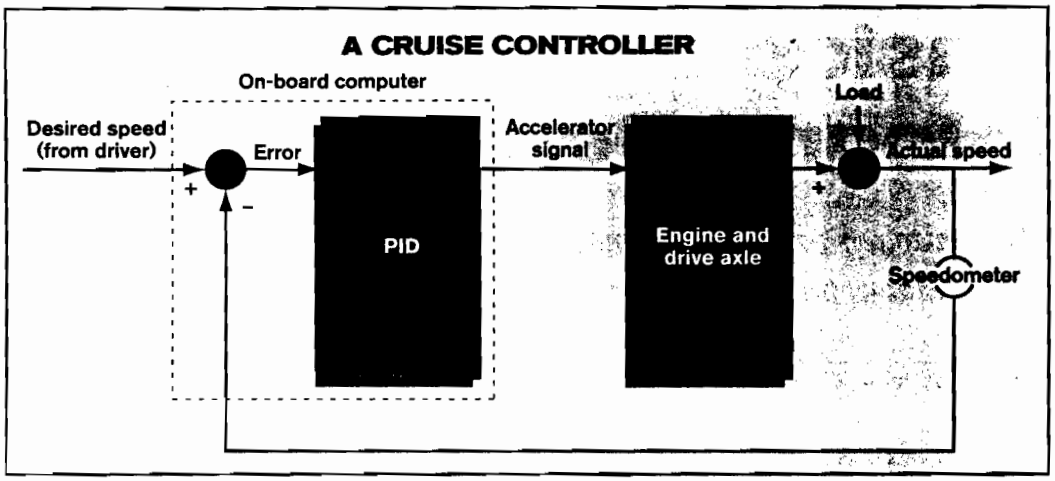

Fig. 2:A cruise controller attempts to minimize errors between the desired speed set by the driver and the car's actual speed measured by the speedometer. The controller detects a speed error when the desired speed is increased or when an added load (such as an uphill climb) slows the car.

some time or the error is changing rapidly, the controller will attempt to make a large correction by generating a large output. Conversely, if the process variable has matched the setpoint for some time, the controller will leave well enough alone.

Tuning the controller

Conceptually, that's all there is to a PID controller. The tricky part is *tuning* it; i.e., setting the P,I, and D tuning constants appropriately. The idea is to weight the sum of the proportional, integral, and derivative terms so as to produce a controller output that steadily drives the process variable in the direction required to eliminate the error.

The brute force solution to this problem would be to generate the largest possible output by using the largest possible tuning constants. A controller thus tuned would amplify every error and initiate extremely aggressive efforts to eliminate even the slightest discrepancy between the setpoint and the process variable. However, an overly aggressive controller can actually make matters worse by driving the process variable past the setpoint as it attempts to correct a recent error. In the worst case, the process variable will end up even further away from the setpoint than before.

On the other hand, a PID controller that is tuned to be too conservative may be unable to eliminate one error before the next one appears. A well-tuned controller performs at a level somewhere between those two extremes. It works aggressively to eliminate an error quickly, but without over doing it.

How to best tune a PID controller depends upon how the process responds to the controller's corrective efforts. Processes that react instantly and predictably don't really require feedback at all. A car's headlights, for example, come on as soon as the driver hits the switch. No subsequent corrections are required to achieve the desired illumination.

On the other hand, the car's cruise controller cannot accelerate the car to the desired cruising speed so quickly. Because of friction and the car's inertia, there is always a delay between the time that the cruise controller activates the accelerator and the time that the car's speed reaches the setpoint. A PID controller must be tuned to account for such *lags*.

PID in action

Consider a sluggish process with a relatively long lag—an overloaded car with an undersized engine, for example. Such a process tends to respond slowly to the controller's efforts. If the process variable should suddenly begin to differ from the setpoint, the controller's immediate reaction will be determined primarily by the actions of the derivative term in equation [1]. This will cause the controller to initiate a burst of corrective efforts the instant the error changes from zero. A cruise controller with derivative action would kick in when the car encounters an uphill climb and suddenly begins to slow down. The change in speed would also initiate the proportional action that keeps the controller's output going until the error is eliminated. After a while, the integral term will also begin to contribute to the con-

THE PID EQUATION

In theory...

$$CO(t) = P\cdot e(t) + I\cdot\left(\int e(t)dt\right) + D\cdot\left(\frac{d}{dt}e(t)\right) \quad (1)$$

In practice...

$$CO(t) = P\cdot\left[e(t) + \frac{1}{T_I}\cdot\left(\int e(t)dt\right) - T_D\cdot\left(\frac{d}{dt}PV(t)\right)\right] \quad (6)$$

Equations [1] and [6]–Both forms of the PID algorithm generate an output **CO(t)** *according to recent values of the setpoint* **SP(t)**, *the process variable* **PV(t)**, *and the error between them* **e(t) = SP(t) – PV(t)**

I is the integral tuning constant

A TYPICAL PROCESS MODEL

$$PV(t) = K \cdot CO(t-d) - T \cdot \left(\frac{d}{dt} PV(t)\right) + LV(t) \quad (2)$$

Equation [2]–The process variable **PV(t)** *is a function of its own derivative plus an earlier output from the controller* **CO(t–d)** *and a random load variable* **LV(t)**. *A PID controller for this process can be tuned according to the values of the parameters* **K**, **T**, *and* **d**.

troller's output as the error accumulates over time. In fact, the integral action will eventually come to dominate the output signal since the error decreases so slowly in a sluggish process. Even after the error has been eliminated, the controller will continue to generate an output based on the history of errors that have been accumulating in the controller's integrator. The process variable may then *overshoot* the setpoint, causing an error in the opposite direction.

If the integral tuning constant is not too large, this subsequent error will be smaller than the original, and the integral action will begin to diminish as negative errors are added to the history of positive ones. This whole operation may then repeat several times until both the error and the accumulated error are eliminated. Meanwhile, the derivative term will continue to add its share to the controller output based on the derivative of the oscillating error signal. The proportional action, too, will come and go as the error waxes and wanes.

Now suppose the process has very little lag so that it responds quickly to the controller's efforts. The integral term in equation [1] will not play as dominant a role in the controller's output since the errors will be so short lived. On the other hand, the derivative action will tend to be larger since the error changes rapidly in the absence of long lags.

Clearly, the relative importance of each term in the controller's output depends on the behavior of the controlled process. Determining the best mix suitable for a particular application is the essence of controller tuning.

For the sluggish process, a large value for the derivative tuning constant D might be advisable to accelerate the controller's reaction to an error that appears suddenly. For the fast-acting process, however, an equally large value for D might cause the controller's output to fluctuate wildly as every change in the error (including extraneous changes caused by measurement noise) is amplified by the controller's derivative action.

Three tuning techniques

There are basically three schools of thought on how to select P, I, and D values to achieve an acceptable level of controller performance. The first method is simple trial-and-error—tweak the tuning constants and watch the controller handle the next error. If it can eliminate the error in a timely fashion, quit. If it proves to be too conservative or too aggressive, increase or decrease one or more of the tuning constants.

Experienced control engineers seem to know just how much proportional, integral, and derivative action to add or subtract to correct the performance of a poorly tuned controller. Unfortunately, intuitive tuning procedures can be difficult to develop since a change in one tuning constant tends to affect the performance of all three terms in the controller's output. For example, turning down the integral action reduces overshoot. This in turn slows the rate of change of the error and thus reduces the derivative action as well.

The analytical approach to the tuning problem is more rigorous. It involves a mathematical *model* of the process that relates the current value of the process variable to its current rate of change plus a history of the controller's output. Random influences on the process variable from sources other than the controller can all be lumped into a load variable LV(t). See equation [2]. This particular model describes a process with a *gain* of K, a *time constant* of T, and a *deadtime* of d. The process gain represents the magnitude of the controller's effect on the process variable. A large value of K corresponds to a process that amplifies small control efforts into large changes in the

D is the derivative tuning constant

A PROCESS MODEL FOR A CAR WITH CRUISE CONTROL

$$F_{engine}(t) - F_{friction}(t) - F_{load}(t) = m \cdot a(t) \quad (3)$$

$$K_{engine} \cdot CO(t) - K_{friction} \cdot v(t) - F_{load}(t) = m \cdot \left(\frac{d}{dt} v(t) \right) \quad (4)$$

$$v(t) = K \cdot CO(t) - T \cdot \frac{d}{dt} v(t) + LV(t) \quad (5)$$

Equations [3], [4], and [5]—The car's velocity **v(t)** *is a function of the cruise controller's output* **CO(t)**, *the car's current acceleration* **a(t)**, *and a load variable* **LV(t)**. *The load variable represents forces on the car from sources other than the engine and friction.*

process variable.

Time constant T in equation [2] represents the severity of the process lag. A large value of T corresponds to a long lag in a sluggish process. The deadtime d represents another kind of delay present in many processes where the controller's sensor is located some distance from its actuator. The time required for the actuator's effects to reach the sensor is the deadtime. During that interval, the process variable does not respond at all to the actuator's activity. Only after the deadtime has elapsed does the lag time begin.

In the thermostat example, the ductwork between the air conditioner and the thermostat causes a deadtime since each slug of cool air takes time to travel the duct's length. The room temperature will not begin to drop at all until the first slug of cool air emerges from the duct.

Other characteristics of process behavior can be factored into a process model, but equation [2] is one of the simplest and most widely used. It applies to any process with a process variable that changes in proportion to its current value. For example, a car of mass m accelerates when its cruise controller calls for the engine to apply a force $F_{engine}(t)$ to the drive axle. However, that acceleration a(t) is opposed by frictional forces $F_{friction}(t)$ that are proportional to the car's current velocity v(t) by a factor of $K_{friction}$. If all other forces impeding the car's acceleration are lumped into $F_{load}(t)$ and the force applied by the engine $F_{engine}(t)$ is proportional to the controller's output by a factor of K_{engine}, then v(t) will obey equation [2] as shown in equations [3] through [5].

In equation [5], the process variable is v(t) and the load variable is $LV(t) = -F_{load}(t)/K_{friction}$. The process gain is $K = K_{engine}/K_{friction}$ and the process time constant is $T = m/K_{friction}$. In this example there is no deadtime since the speed of the car begins to change as soon as the cruise controller activates the accelerator. The car will not reach its *final* speed for some time, but it will *begin* to accelerate almost immediately.

If a model like [2] or [5] can be defined for a process, its behavior can be quantified by analyzing the model's parameters. A model's parameters in turn dictate the tuning constants required to modify behavior of a process with a feedback controller. There are literally hundreds of analytical techniques for translating model parameters into tuning constants. Each approach uses a different model, different controller objectives, and different mathematical tools.

The third approach to the tuning problem is something of a compromise between purely heuristic trial-and-error techniques and the more rigorous analytical techniques. It was originally proposed in 1942 by John G. Ziegler and Nathaniel B. Nichols of Taylor Instruments and remains popular today because of its simplicity and its applicability to any process governed by a model in the form of equation [2]. The Ziegler-Nichols tuning technique will be the subject of "Back to Basics" (*CE*, Aug. 1998).

Application issues

Experienced PID users will note that none of the discussion so far applies directly to the commercial PID controllers currently running more than 90% of their industrial processes. Several sub-

Tuning a controller entails choosing appropriate values for P, I, and D.

tle flaws in the basic PID theory have been discovered during the last 50 years of real-life applications.

Consider, for example, the effects of *actuator saturation*. This occurs when the output signal generated by the controller exceeds the capacity of the actuator. In the cruise control example above, the PID formula may at some point call for a million 16 lbf-ft torque to be applied to the drive axle. Mathematically, at least, that much force may be required to achieve a particularly rapid acceleration.

Of course real engines can only apply a small fraction of that force, so the actual effects of the controller's output will be limited to whatever the engine can do at full throttle. The immediate result is a rate of acceleration much lower than expected since the engine is "saturated" at its maximum capacity.

However, it is the long-term consequences of actuator saturation that have necessitated a fix for equation [1] known as *antiwindup protection*. The controller's integral term is said to "wind up" whenever the error signal is stuck in either positive or negative territory, as in this example. That causes the integral action to grow larger and larger as the error accumulates over time. The resulting control effort also keeps growing larger and larger until the error finally changes sign and the accumulated error begins to diminish.

Unfortunately, a saturated actuator may be unable to reverse the error. The engine may not be able to accelerate the car to the desired velocity, so the error between the desired velocity and the actual velocity may remain positive forever. Even if the actual velocity does finally exceed the setpoint, the accumulated error will be so large by then that the controller will continue to generate a very large corrective effort. By the time enough negative errors have been accumulated to bring the integral term back to zero, the controller may well have caused the car's velocity to overshoot the setpoint by a wide margin.

The fix to this problem is to prevent integrator wind-up in the first place. When an actuator saturates, the controller's integral action must be artificially limited until the error signal changes sign. The simplest approach is to hold the integral term at "s last value when saturation is detected.

Alternative implementations

The PID formula itself has also been modified. Several variations on equation [1] have been developed for commercial PID controllers; the most common being equation [6]. This version involves differentiating the process variable PV(t) rather than the error e(t) = SP(t) – PV(t). The idea here is to prevent abrupt changes in the controller's output every time the setpoint changes. Note that the results are the same when the setpoint SP(t) is constant.

The tuning constants in equation [6] differ from those in equation [1] as well. The controller's proportional gain now applies to all three terms rather than just the error e(t). This allows the overall "strength" of the controller to be increased or decreased by manipulating just P (or its inverse).

Several subtle flaws in the basic PID theory have been discovered during the last 50 years of real-life applications.

The other two tuning constants in equation [6] have been modified so that they may both be expressed in units of time. This also gives some physical significance to the *integral time* T_I. Note that if the error e(t) could somehow be held constant, the total integral action would increase to the level of the proportional action in exactly T_I seconds. Although the error should never remain constant while the controller is working, this formulation does give the user a feel for the relative strengths of the integral and proportional terms; i.e., a long integral time implies a relatively weak integral action, and vice versa. □

For more details on the practical issues of applying PID controllers to real-life control problems, circle 209 or refer to "Process Control Systems" by F. Greg Shinskey, available from the Foxboro Training Institute at 1-888-FOXBORO. The author gratefully acknowledges Mr. Shinskey's assistance in the preparation of this article.

Was this article...

Interesting?	Useful?	Not useful?
390	391	392

Comments?
Send e-mail to Vance VanDoren at controleng@juno.com.